LeGEND | Digital Random Number Generator

Anthony Shen
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Champaign, IL
arshen2@illinois.edu

David Phillips
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Champaign, IL
daviddp2@illinois.edu

Michael Gamota
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Champaign, IL
mgamota2@illinois.edu

Rohan Rughani
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Champaign, IL
rughani3@illinois.edu

Rahul Ramanathan Krishnamoorthy
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Champaign, IL
rahulr9@illinois.edu

Jay Nathan
Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Champaign, IL
jayrn2@illinois.edu

***Abstract*—Random number generators have been a staple in modern computing systems because of their various use cases: generating secure encryption keys or initialization vectors, providing truly random inputs for simulation, and many more. However computing systems are by nature deterministic, as a result, producing quality random numbers that are statistically independent, uniformly distributed and unpredictable is much more difficult than it might seem. Pseudo Random Number Generators (PRNGs) are widely used in tandem with a random seed to produce an output which seems random. Nevertheless, PRNGs tend to have a critical pitfall: they are deterministic in nature and if the seed is discovered then the random numbers can be recalculated. For purposes where the vulnerabilities of the PRNGs are a problem to be avoided, True Random Number Generators (TRNGs) are utilized. Instead of using a mathematical model to deterministically generate numbers that look random, a TRNG extracts entropy from a wide variety of physical sources (a.k.a entropy source). Unlike PRNGs, TRNG sampling can pose an issue for present day computing systems as they typically require long delay times and device I/O. An approach to improving TRNG's performance is to combine the functionality of both a TRNG and PRNG in a cascade fashion, where the seeds of the latter are provided randomly by the TRNG. This combination is termed Digital Random Number Generator (DRNG) by Intel and is the scheme that the proposed mixed IC design aims to satisfy. This design leverages 32 calibratable metastable latches, and 32 clock jitter ring oscillators in tandem with hardware entropy testing mechanisms, NIST certified post-processing conditioner, area efficient PRNG a.k.a Trivium, and a NIST certified PRNG, along with an output buffer to generate random numbers at a fast and scalable rate.**

I. Project Summary

A. Main Features and High-Level Architecture

Our chip is a digital random number generator (DRNG) designed to interface with an x86 CPU. It can fulfill RDRAND and RDSEED requests for sizes of 16,32, and 64 bits. The chip is designed for operation at a 200 MHz system clock and outputs up to 16 bits per cycle at 100 MHz. These random numbers are generated using physical entropy sources as seed values. We use two arrays of 32 entropy sources based on two different random physical processes: nondeterministic clock jitter and thermal noise. These are referred to as jitter sources and latch sources. Each source samples one bit per clock cycle. Since both entropy sources are influenced by temperature fluctuations, there are additional temperature sensors on chip to invalidate biased bit generation. An ”Online Health Test” (OHT), which monitors consecutive bit repetition and distribution of bits over samples size of 1024, is responsible for deactivating broken entropy sources and calibrating the latch sources. Jitter sources cannot be calibrated. The random bits generated by each entropy source are stored in a 2kx32 dual-

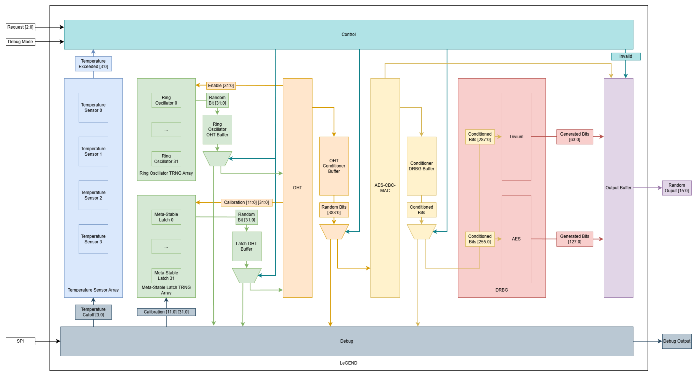

Fig. 1. Block Diagram of the LeGEND Architecture

port SRAM. The read port is fed into a bit compactor which provides the AES-CBC-MAC conditioner with the key and seed values needed to process future random inputs. The conditioned outputs are fed into the two PRNGs: Trivium, which is an area efficient cipherstream generator that is doubling as a PRNG and can supply a near endless stream of random bits without stalling, and the AES-CTR based PRNG which is NIST certified. The output of the conditioner and PRNGs are then stored in an output buffer which runs at approximately 100MHz for signal integrity on the output pins. Additionally, for debug and state functionality, there are registers which are controlled through the SPI interface on-chip. Depending on the state of the registers, certain buffers are read into and out from on the chip to provide module level granularity and testing.

B. Physical Design

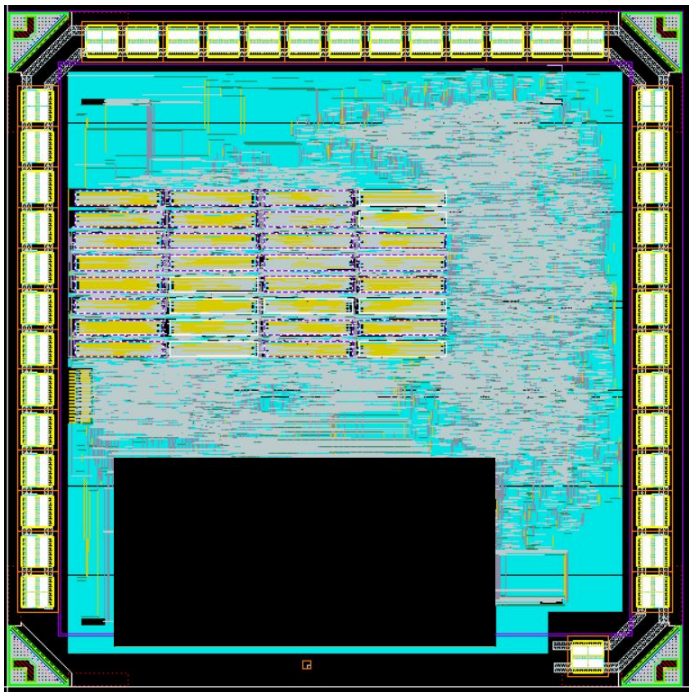

Fig. 2. Final placed and routed design (with AP and power mesh layers disabled)

The final physical implementation of the chip targets a 1.2 V core supply and occupies 574,699.3 µm² of active area. Timing closure on the routed netlist supports operation up to 220 MHz, enabling a peak output rate of 352 MB/s at 220 MHz. For the final floorplan, the digital components (excluding SRAM) were auto placed and routed and the position of the SRAM was fixed. Reserved regions were introduced for the entropy-source and temperature sensors to ensure deterministic placement and prevent digital routing congestion from encroaching into analog keep-out regions. This approach produced a stable placement solution that could be reproduced across multiple PnR runs while still allowing iterative timing optimization to eliminate residual hold violations without compromising setup slack.

Mixed signal regions were partitioned into 32 individual latch sites placed as a 8x4 array, one reserved region for the full set of 32 jitter sources, and four temperature sensor regions near the corners of the core to provide good coverage for thermal tamper detection. The analog components were then manually placed and routed post digital component PnR. Pin access points were preplanned such that the analog placement aligned with the routed digital pin locations. Final assembly was validated with sign-off checks on the merged layout to confirm geometric correctness and connectivity (DRC/LVS).

The layout was optimized by removing the bottom row of the IO ring to expand the total core area and mitigate routing congestion. Power integrity was improved by providing two VDD/VSS pairs, with one VDD-VSS pair on the left side of the chip and one on the right side, ensuring more uniform IR drop and robust distribution across the power mesh. The device provides random data through the `rand_byte_io[15:0]` port. A request/response handshake is implemented via `rand_req_io` (asserted during an external request) and `rand_valid_io` (asserted when output data is available), with `rand_req_type_io[2:0]` encoding the specific request type and width. System control is managed by `rst_n_io` as the global reset and `ic_clk_io` as the primary system clock. A standard SPI interface comprising `mosi_io`, `miso_io`, `spi_data_ready_io`, and `ss_n_io` allows for the configuration of internal registers for debugging and module-level control during validation. Finally, `debug_io` enables the chip's debug mode, while `temp_debug_io` allows the temperature sensors to be disabled for directed bring-up and isolation testing.

The final implementation phase yielded two key insights. First, placing the latch array in a high-density area improved the connection to the digital test and storage logic, but it also crowded the wiring at the interface where digital and analog circuits meet. This congestion made the physical integration of the analog parts more difficult, as finding space for pins and managing wire traffic became a challenge. Second, while manually placing the analog components helped align them with the intended pin locations, future revisions would benefit from packaging the analog blocks as standard cells. Using these pre-packaged blocks would allow automated design tools to handle placement, pin routing, and power connections more reliably.

C. Design Choices and Justification

Originally there were 128 total entropy sources for random bit generation. However, this project consistently faced area constraints, and to meet a compromise between area and performance the total number of entropy sources was reduced to 64, 32 jitter and 32 latch. Therfore, the SRAM was also reduced in area and there was ample real estate available on the chip for entropy source integration and routing.

The OHT also contains a streaming bit compactor to ensure that valid random bits are inputted into the conditioner unit. Although the streaming bit compactor is pipelined, there was a tradeoff between the latency and parallel comparisons it could do with a valid mask. To minimize the delay between multiple SRAM reads and the amount of bits needed to initialize the conditioner, the streaming bit compactor did 16-bit comparisons and shifted valid bits at the end and discarded upper bits with an overflow buffer implemented to maintain random bits generated.

Furthermore, the AES-CBC-MAC conditioner is not a design which can be pipelined since the inputs are dependent on the outputs. Several tradeoffs were taken into consideration, such as creating parallel conditioners but the area increase for multiple conditioner units was concluded to come at a greater cost than the performance gains.

There are two PRNGs in this design to utilize more sources of randomness. Since area constraints were always an issue, Trivium was implemented as one of the PRNGs. It promised the same behavior as other PRNGs at a lower area cost. We also wanted this chip to meet a specification, since the conditioner and OHT were listed in the specification provided by the National Institute of Science and Technology (NIST) for DRNGs we also implemented the AES-CTR PRNG. Although Trivium is an area efficient design, it comes at a cost of several thousand cycles of latency. The secondary PRNG was implemented to generate random bits for that duration.

Since signal integrity is not guaranteed at higher frequencies past 100MHz, the chip uses an output buffer which ensures that the output pins are driven at maximum of 100MHz. The output buffer also acts as a temporary storage for the outputs of the conditioner and PRNGs to instantly serve random bit requests.

D. Verification and Test Strategy

1) Digital Components: The Digital Verification (DV) work on our design followed a bottom-up approach. Individual members were responsible for establishing an abstraction layer (choosing the inputs, outputs, and timing behavior) for each of their modules, writing the RTL for that module, and then exhaustively verifying the functionality of that module using functional verification. Only after these steps were done was a component allowed to be integrated into the overall design. As a result, all of the major digital components have their own associated test benches and make commands within our repository.

If modules were able to be tested against some kind of software reference implementation (such as the two AES implementations in our design), golden vectors generated via software (Python, C, etc.) were used as inputs instead of random numbers. Passing verification against those reference implementations gave us confidence that we had not incorrectly implemented those algorithms and is a superior method over a custom test-bench.

After each module was cleared and integrated, we created several "top-level" test benches that exhaustively test the behavior of the chip by monitoring the I/O and inputting test vectors to the pins. We created tests for the Behavioral, Post-Synthesis, and Gate-Level models of our design (however Post-Synthesis testing proved to be futile due to uncorrected hold-violations pre-PNR).

Additionally, the NIST provides open source DRNG verification testing to ensure that the bits generated by the DRNG are truly random and can be utilized. This open source test bench was integrated into the design and the conditioner and PRNGs were verified by running the outputs of each module with the provided test benches. Of the 15 test cases provided by NIST, the conditioner and PRNGs pass 14. The missing test case is most likely due to a software bug in the provided tests and was not something to be concerned about since it was a redundant test case.

2) Analog Components: The Analog Verification work followed a similar bottom-up approach to the DV work. Each individual module was tested in a purely analog simulation to verify functionality across various corners and conditions. For the random number generators, this means looking at how various conditions impacted the distribution of 1's and 0's were impacted. For the temperature sensor, the relationship between oscillations and temperature was verified.

After verification of the analog performance, mixed-simulations were performed to verify the interface between the digital and analog components. This mainly focused on the OHT's ability to calibrate the strong-arm latch TRNG and to restart the ring oscillator TRNG in the event that the output was skewed.

With schematic level verification completed, each analog module was laid out. Then post-layout simulation was performed to verify functionality with extra non-ideal capacitance and resistance from the layout. In the strong-strong arm latch, this meant balancing the 2 sides of the latch as closely as possible. For the ring oscillators, the simulation needed to verify its ability to start oscillating.

E. Comparison with Proposal

The original proposal for this project was a secure cryptographic processor with an ADC for secure real time applications. However, after further investigation and certain implementations of modules required for the project, it was realized that this project was not feasible under the given area constraints. As a result, this project was turned into a different application of computer security: random bit generation. We were able to recycle certain modules originally implemented for the cryptographic processor, and still meet the proposed timeline. Additionally, the analog components became an integral part of the design and the reliability of their functionality was also core to the design. Therefore, the mixed signal aspect of the project was further emphasized by this switch.

II. Individual Contributions

- **David**
  - Conditioner
  - Output Buffer.
  - Top Level Integration
  - Digital Verification
- **Rohan**

    * Entropy Validation
    * Hardware Entropy Source Calibration & Validation
    * SRAM
    * Trivium PRNG
  - **Jay**
    * AES DBRG
    * DFT & GLS
    * Top Level Integration
    * Layout Debugging
  - **Rahul**
    * PD flow
    * PNR Scripting
    * Layout Debugging
  - **Michael**
    * StrongARM Latch Entropy Source
    * Control/Debug Architecture/RTL
    * SPI Interface
    * Top level integration
  - **Anthony**
    * Temperature Sensor
    * Ring Oscillator TRNG
    * Physical Design
    * Layout Debugging

III. Major Challenges

The initial project proposal included a specification for a secure hardware processor with an analog to digital converter for mission critical processes. After a few weeks of design and research, the area constraint of this project became a limiting factor as Montgomery modular multiplier modules were necessary for a fast and secure encryption algorithm implementation. The digital portion of this project was scrapped and behind for a few weeks due to that oversight, however one of the PRNGs: Trivium was reusable for the purposes of the new DRNG.

Another challenge encountered during the development of this project was due to the size of the sampling window needed for each entropy source. NIST specifications have a minimum number of 1024 bits that are needed for each entropy source to validate the randomness that is generated. The area constraint was another consideration here since the SRAM would occupy 50 percent of the chip area if this design was implemented. The earlier versions of this design proposed 128 entropy sources split evenly among the latch and jitter sources, but a reduction to 64 sources was made along with implementing a dual port SRAM to address this constraint. Furthermore, the conditioner module was initially designed to be pipelined, but the CBC-MAC algorithm restricts the implementation in this design to remain multi-cycle instead.

IV. Module Descriptions and Documentation

A. Analog Components

Metastable Latch Entropy Sources [1] - The metastable latch entropy source is a StrongArm latch which uses a digitally switched binary sized transistor array for calibration instead of the original differential pair input. When the clock is low, the circuit charges up, then when the clock goes high, there is a race between the positive and negative sides of the latch to drain first. The value of the output depends on which sides drains first, which is determined by noise in a perfectly balanced/calibrated latch. The motivation for this is to be able to easily calibrate out the input offset error that arises from process variation and any layout imbalance. The overall design includes a cross coupled inverter pair, the transistor array, output buffers (TSMC standard cell), output flip flops (TSMC standard cell), and a few transistors to clock the circuit. Since this module can be monitored and calibrated at startup and runtime, we connect it to an Online Health Test module. The layout for this module includes over a hundred dummy transistors and an interdigitized layout for transistors with multiple fingers.

Clock Jitter Entropy Sources - The clock jitter entropy sources generate random bits utilizing the jitter from the clock and ring oscillator. In this implementation, two differently sized ring oscillators are sampled by the clock and XORed together. To sample the ring osciallator, a flip-flop is used with the ring oscillator as the input. Due to thermal noise and device mismatches, the exact timing of each clock edge varies slightly. Two ring oscillators are used to accumulate more jitter. The output of each entropy source is monitored through the OHT and can be disabled if the output is skewed past the acceptable limit.

Temperature Sensors - The temperature sensor is used to detect heat tampering attempts on the chip. If the temperature exceeds the cutoff, the chip will stop outputting until the temperature is below the cutoff. This sensor uses a ring oscillator that has been tuned to have a linear relation between the temperature and frequency. The chip will sample the ring oscillator over a period of time and then check to see if the number of oscillations exceeds the cutoff. The delay of an inverter increases as temperature increases so if the oscillation count is below the cutoff, that means the temperature of the chip has exceeded the acceptable limit.

B. Digital Components

Control - The control unit is responsible for setting the operation mode (debug or normal). The control unit also includes the SPI interface which is capable of reading and writing registers that set the temperature sensor thresholds and allow disabling of certain modules. When in debug mode, the SPI interface is used to configure debugging crossbars to allow fine-grained access to individual entropy sources, module bypass, and other features useful for debugging.

Online Health Tests (OHT) - Serve to verify the entropy sources to ensure that their outputs are truly random and that nothing external is tampering with their results. If any tampering or low enough entropy is detected then the output of those sources are not considered for any random number generation. For the strongARM latch sources, there are tailored health tests that detect failure and calibrate the entropy source, in this case the metastable latch has calibration and there are different tests for those. NIST specifies standards that the OHT has to meet, there are tests to meet each of these standards: if a consecutive number of 1s or 0s are detected then the entropy source needs to restart or be considered as a failure and the output should not be used. Additionally, if the OHT detects that the entropy within a minimum sampling window is below a certain threshold dictated by the entropy source parameters, then that source is considered a bad entropy source and its outputs are not considered.

SRAM - The SRAM is used to store the bits from each entropy source for a 1024-bit verification window. It is a 2048x32 dual-port pipelined SRAM with one port permanently writing and the other port permanently reading. Since the SRAM was generated from an IP, it was verified through LVS checks and functionality simulations with certain modifications to the provided files to disregard timing constraints. Additionally, it was ensured that the read and write address would never be the same in this design to address timing concerns which arose from address collisions in the SRAM.

Streaming Bit Compactor - This chip was designed to write 32 bits of the latch entropy sources into the SRAM at even address locations, and 32 bits of the jitter entropy sources into the odd address locations. The consideration of whether these bits are being generated from a valid entropy source are taken into account during the read. There is a mask which is layered onto the 32 bits read from an address, but there could be a scenario where not all 32 bits are valid and certain bits need to be discarded. In order to address this, a 32-bit shifter is created which sorts the invalid bits into the most significant bits and only stores 16-bit chunks from the lower portions. This shifter can heavily impact the performance of the IC as each bit generated by the entropy sources needs additional time to be verified before being written into the SRAM. So an overflow buffer is also implemented to ensure that when the 16-bit chunks are filled up, any left over valid bits are stored into a buffer and utilized in the next 16-bit chunk. The 16-bit chunks were then fed into a FIFO which stored the required number of bits for the conditioner seed and initialization vector.

Conditioner - The NIST states in their various recommendations for CSPRNGS that they should contain some kind of "conditioning" module that employs a cryptographic algorithm not for the purpose of encrypting data, but instead to distill and mix bits around in an entropy string and improve its quality. In our design, the algorithm chosen is AES_CBC_MAC due to its relative simplicity and is the conditioner of choice in Intel's designs. It ingests a 256-bit message and a 128-bit key, and outputs a 256 MAC typically used for authentication but in this use case is just a superior random bitstring. The design utilizes a FSM-based control unit and time-multiplexed execution hardware to minimize the area footprint. From the assertion of the "start" signal, the module has an execution time of 30 clock cycles. A wrapper is used in the top level to facilitate basic valid/ready handshakes with the OHT and DRBGs.

AES CTR DRBG - The AES CTR DRBG is a deterministic random bit generator fully compliant with NIST SP 800-90A. It expands an incoming 256-bit seed from the Conditioner into a continuous stream of 128-bit random blocks. Internally, it manages a secret AES-128 key and a 128-bit counter (V). Generation involves encrypting the counter to produce output, followed immediately by a post-generate update phase that evolves the internal state to ensure forward secrecy. The core is encapsulated in a Wrapper (drbg_wrapper) that integrates the module into the system. This wrapper manages the Reseed Interval (forcing a fresh seed from the Conditioner every 511 blocks), handles ready/valid handshaking to support system backpressure, and includes a serial debug interface that allows injecting specific seeds for deterministic silicon validation.

Trivium DRBG [2] - The Trivium DRBG is a random bit generator algorithm which is extracted from academic literature. The goal of this particular random bit generator is to minimize the area as much as possible without decreasing any of the corresponding security or randomness of a DRBG. As a result, Trivium only takes up approximately 4% of the total area provided but has the capability to stream 8 bytes in a cycle for higher throughput randomness. The low hardware cost comes from the simplicity of using shift registers and the properties of the XOR gate for producing pseudo random values as long as a warm-up time and random seed are provided.

Output Buffer - The output buffer is a module that delivers the random numbers generated by our device to a debugging FPGA or similar device. In a real CPU, the bus connecting the CSPRNG to the host CPU would most likely be 64 wires wide in order to minimize latency when returning random numbers. In our ASIC it is infeasible to dedicate 64 pins to solely output, so we must slice the numbers and send them out in 16-bit chunks using the Output Buffer. Since we are interfacing directly with a FPGA with a connector likely not designed for our target frequency the output buffer divides the base clock by 2 in order to achieve a much more manageable 100 mHz output frequency. This restricts our device's theoretical maximum possible throughput to 160 MB/s, but it must be done for signal integrity. This module also handles valid/ready handshaking with the DRBGs and Conditioner.

V. Testing Instructions

A. *Top Level Digital Testing and Verification*

Testing commands for the entire digital design can be found in the main directory in /groups/ece427-group1/ECE427_root/LeSCOPE/rtl. Running "make top" in the rtl directory will execute a self-checking behavioral test-bench that exhaustively tests all debug scenarios as well as standard operating behavior. Running "make post_pnr" will run the same test on the gate-level model of our digital design.

B. *Analog Components*

Testing commands for individual Analog components should be done in /groups/ece427-

group1/ECE427_root/legend_midterm.

Metastable Latch Entropy Sources - To test a latch entropy source, say source #28 with the calibration bits set to 1100_0011_1100_0000, the debug pin would be pulled high and the following SPI messages would be sent:

	data[21]	data[20]	data[16:19]	data[15:0]			
	Mux Select/Register Op	Read/Write	Address	Payload			
	1-Register Op	1-Write	00111 - Calibration bits	11000011 11000100			
	data[21]	**data[20:19]**	**data[18:14]**	**data[13:12]**	**data[11:7]**	**data[6:5]**	**data[4:0]**
Then monitor the output	Mux Select/Register Op (0-Mux select)	001 - Latch RNG 0-31 to output pin 2	11100 - Index 28	000 - VCC/CLK/Conditioner/DRBG to output pin 1	00001 - CLK	000 - No external input	0 - No external input

Fig. 3. Example of a SPI Message

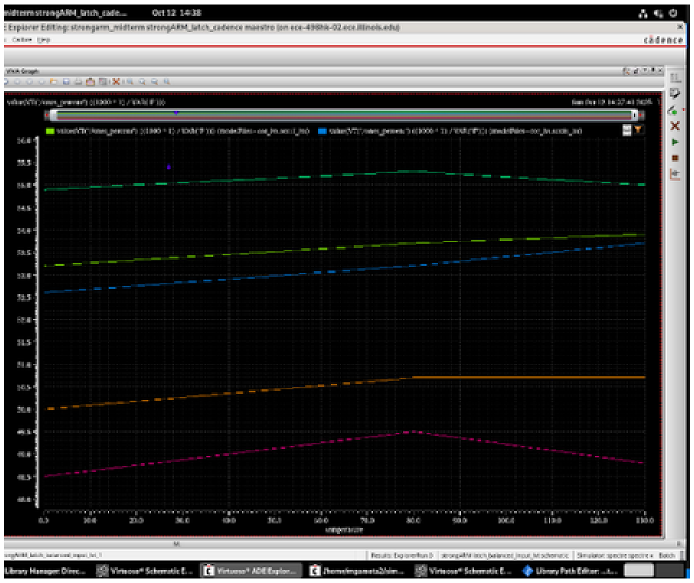

Fig. 4. This graph shows the percentage of 1s present in a 1000 bit sample of StrongArm latch outputs.

Clock Jitter Entropy Sources - To test a jitter TRNG, open the maestro view of ro_trng_v2. In design variables nw_1, pw_1, nw_2, pw_2, and jitter. After running the graph will display a plot of ones and zeros. In addition, RO_OUT is the actual output stream of the jitter entropy source. Path to Folder: /groups/ece427-group1/ECE427_root/legend_midterm/ring_oscillators_cadence

Temperature Sensors - To test a temperature sensor, open the maestro view of ro_temp_v2 and set the temperatures in the design variables and set the TEMP_CMP[13:0] in schematic. After running the graph will display TEMP_IN[13:0] which will be the digitized temperature output and temp_cmp_out which will tell if the temperature exceeded the cutoff. Path to Folder: /groups/ece427-group1/ECE427_root/legend_midterm/ring_oscillators_cadence

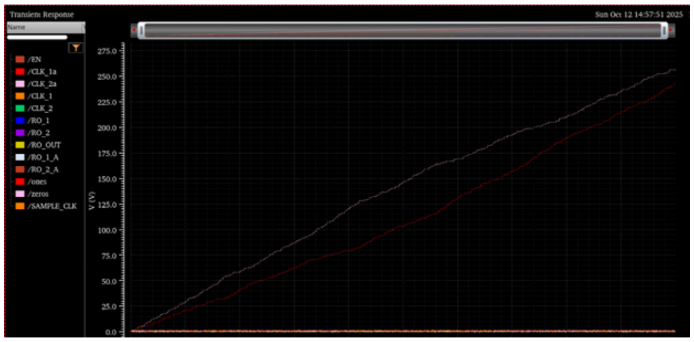

Fig. 5. Output of Clock Jitter Test

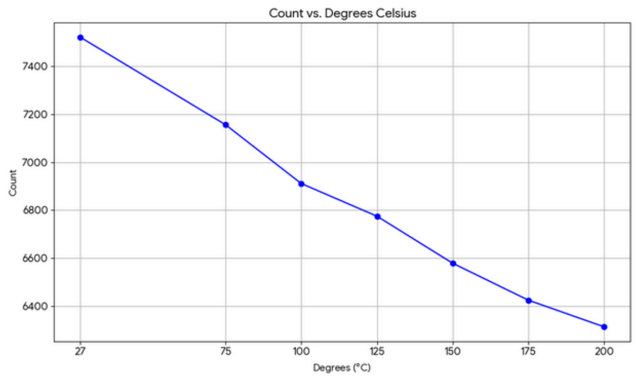

Fig. 6. Output of Temperature Sensor Test

C. Digital Components

Testing commands for Digital modules should be done in /groups/ece427-group1/ECE427_root/LeSCOPE/legend_midterm/rtl.

OHT - The OHT module consists of a lot of submodules; however, the functionality of each submodule is identical. To verify the submodule, a mixed simulation was created to test the calibration and verification of the entropy source. In the Cadence folder for the StrongARM latch there is a Maestro file which outputs the calibration and illustrates the entropy verification of the OHT module. The good entropy signal outputted from the OHT module indicates that the entropy passes the 1024-bit sampling window and the other OHT test cases for entropy verification. To run the mixed simulation, simply open the “mixed_sim” cell’s Maestro file and click the green play button to start a 10 µs simulation demonstrating similar behavior. Additionally, there is a directed test for the overall top level module for the verification of the SRAM and bit packer needed to output to the conditioner. The results of the testbench can be verified in verdi by running “make oht_top” and then “make oht_top_verdi”

Conditioner - This module has a self-checking test based on a Python library implementing the same algorithm (AES_CBC from PyCryptodome). Simply run “make aes_cbc_mac” in the /groups/ece427-group1/ECE427_root/legend_midterm/rtl/ directory and 1000 golden vectors will be compared to the AES_CBC_MAC output. More random vectors can be tested by re-running the /groups/ece427-group1/ECE427_root/LeSCOPE/bin/aes_cbc_mac.py script (after sourcing aes_env/bin/activate to import the proper libraries) to re-generate golden vectors.

AES_CTR_DRBG - Implemented in hdl/aes_ctr_drbg.sv, this is the main DRBG core, which strictly follows the NIST SP 800-90A standard. It takes a 256 bit seed and streams 128 bit random blocks using an AES-128 encryption core, performing a post generate update after every request to ensure forward secrecy. The core is instantiated within drbg_wrapper.sv, which manages the higher level control logic: it buffers the output, enforces the reseed interval (default 511 blocks), and handles the handshake with the Conditioner (which uses AES-CBC-MAC to distill entropy). The testbench hvl/aes_ctr_drbg_tb.sv verifies the core by driving clk/reset and using 3 tasks: do_instantiate(seed), do_generate(N), and do_reseed(conditioner_seed), to inject known seeds, request N blocks, and force reseeds. Since the DRBG is deterministic, we validate functionality statistically: running make drbg produces a large sample of output blocks (e.g., 4000) into drbg_blocks.txt. A Python script then flattens these blocks to bytes and plots a 256-bin histogram to verify uniformity.

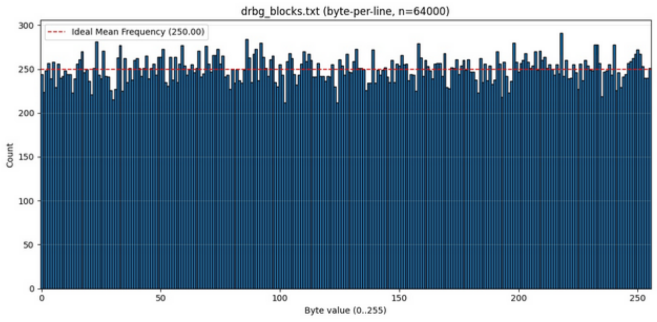

Fig. 7. Output of AES_CTR_DEBUG Test

In figure 7, each byte value clusters around the ideal mean (250 counts for 64kB total) with natural variance and no significant outliers. This confirms that the AES-CTR implementation, including the post-generate update and reseed guard, is producing statistically uniform output as expected.

Trivium - Since Trivium was taken from an academic paper, there was no golden implementation for comparison. Any software implementation could also have a similar misunderstanding that the hardware implementation also contains. Therefore, Trivium is verified by checking the distribution of the output, this can be verified by running “make trivium” in the rtl directory and in the vcs directory a “rrand_output.txt” file is outputted. Afterwards, a python script outside of the server environment was ran on the output .txt file to create a histogram of the distribution of the numbers created from each byte. Figure 8 illustrates the distribution from Trivium with 500k samples.

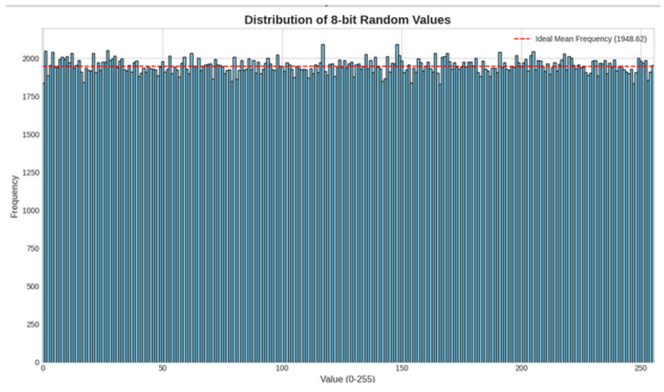

Fig. 8. Output of Trivium Test

VI. Post-Silicon Validation Plan

A. PCB Design

Michael, Rohan, Anthony and Rahul will design the PCB in the beginning of the Spring 2026 semester and have it ready for when our chips return from packaging. It will be a simple design and will likely consist of GPIO pins that correspond to each pin on the ASIC.

B. FPGA Host

In order to emulate the behavior of a host CPU, we will be using an FPGA to drive the inputs of our chip and monitor the outputs. One of our group members has a Spartan-7 FPGA developer kit with enough GPIO pins for all of our IO. We will use Vivado to program the FPGA, as it is a Xilinx part.

C. Validation

The most basic validation test is to drive the inputs and see if random numbers are actually returned. If this stage works, then we will consider the chip functional in its normal state, assuming the numbers returned pass statistical NIST tests.

Should we have issues with the basic behavior, we will resort to using our various debugging SPI registers to modify the behavior of the ASIC to identify bug(s) and attempt to bypass problematic modules.

For more robust validation of our design, we intend to port the top-level testbenches we used during the design phase, modify them slightly to be synthesizeable on the FPGA, and run them as is. If these testbenches work successfully, we can safely assume our debug logic is also correct.

D. Encryption Application

Random number generation is fundamental to secure encryption. Most commonly, random numbers are used as private keys in encryption schemes like RSA. Another application for random number is in challenge-response protocols such as Schnorr Identification protocol. Random numbers are also required to ensure individual privacy in differential privacy schemes. Assuming that at least 1 of the 64 entropy sources is functional, a proof of concept will be demonstrated at the end of the Spring 2026 semester.

Additionally, a similar product to the YubiKey USB drives for secure authentication and verification can be implemented using this IC. If the IC is functionally correct with minimal issues then we would also like to pursue this avenue and create a similar authentication tool with our IC embedded into the design to generate and store authentication vectors.

References

[1] J. Kim and H. Chae, ”A 10-Gb/s True Random Number Generator Using ML-Resistant Middle Square Method,” in *IEEE Journal of Solid-State Circuits*, vol. 59, no. 7, pp. 2321-2329, July 2024, doi: 10.1109/JSSC.2023.3346428.

[2] C. De Cannière and B. Preneel, “Trivium,” in *Lecture Notes in Computer Science*, pp. 244–266, 2008, doi: https://doi.org/10.1007/978-3-540-68351-3_18.

[3] S. Keller and T. A. Hall, “The NIST SP 800-90A Deterministic Random Bit Generator Validation System (DRBGVS),” National Institute of Standards and Technology, Gaithersburg, MD, USA, NISTIR, Oct. 2015.

[4] National Institute of Standards and Technology, “Recommendation for Random Number Generation Using Deterministic Random Bit Generators,” *NIST Special Publication 800-90A*, Rev. 1, Jan. 2012.